Carnegie Mellon
Software Engineering Institute

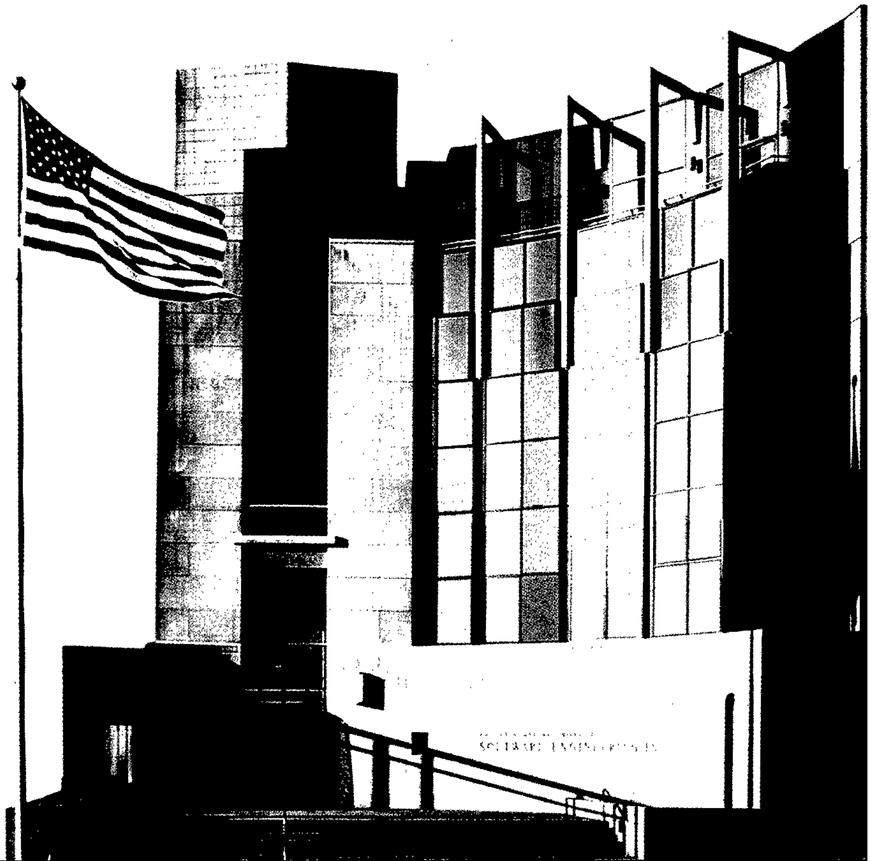
Securing Internet Sessions With Sorbet

Fred Long
Scott Hissam
Robert C. Seacord
John Robert

July 1999

COTS-Based Systems Initiative

Technical Note
CMU/SEI-99-TN-002



Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213-3890

Securing Internet Sessions With Sorbet

Fred Long
Scott Hissam
Robert C. Seacord
John Robert

July 1999

COTS-Based Systems Initiative

Technical Note
CMU/SEI-99-TN-002

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

Please refer to http://www.sei.cmu.edu/publications/pubweb.html for information about ordering paper copies of SEI reports.

Carnegie Mellon University does not discriminate and Carnegie Mellon University is required not to discriminate in admission, employment, or administration of its programs or activities on the basis of race, color, national origin, sex or handicap in violation of Title VI of the Civil Rights Act of 1964, Title IX of the Educational Amendments of 1972 and Section 504 of the Rehabilitation Act of 1973 or other federal, state, or local laws or executive orders.

In addition, Carnegie Mellon University does not discriminate in admission, employment or administration of its programs on the basis of religion, creed, ancestry, belief, age, veteran status, sexual orientation or in violation of federal, state, or local laws or executive orders. However, in the judgment of the Carnegie Mellon Human Relations Commission, the Department of Defense policy of, "Don't ask, don't tell, don't pursue," excludes openly gay, lesbian and bisexual students from receiving ROTC scholarships or serving in the military. Nevertheless, all ROTC classes at Carnegie Mellon University are available to all students.

Inquiries concerning application of these statements should be directed to the Provost, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-6684 or the Vice President for Enrollment, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-2056.

Obtain general information about Carnegie Mellon University by calling (412)-268-2000.

Contents

List of Figures

List of Tables

Abstract

More and more organizations are using intranets, and even the Internet, as the communications media for important data. However, such communications media are inherently insecure and subject to hijacking. To secure these connections, mechanisms must be built on top of the underlying communications facilities. In this paper, we discuss one such security mechanism and describe an implementation using common object request broker architecture (CORBA) -based interceptors.

1 Introduction

There exist today a number of commercial off-the-shelf (COTS) products that can be used to secure communication over an Intranet or the Internet. These products include implementations of the Secure Socket Layer (SSL) as well as other security solutions.

Secure Sockets Layer is a protocol developed by Netscape for transmitting private documents via the Internet. SSL is used by Netscape Communicator and Microsoft Internet Explorer and supported in a wide variety of middleware products—including Object Request Brokers from Inprise, Iona and JavaSoft. As a result, SSL is a strong candidate for securing transactions between objects in a distributed object system. However, existing SSL implementations have some limitations. The SSL Pack from Inprise, for example, is only available through the Java Native Interface (JNI) as platform specific libraries. The use of platform specific libraries limits deployment of the client as an application[1] on platforms for which SSL libraries are not available. Licenses must also be acquired in this case for each client platform.

As a result of these and other limitations, the COTS-Based Systems Initiative at the Software Engineering Institute developed a lightweight, pure Java-based security solution that could be easily deployed as a Java orblet, application, or servlet. Secure ORB enterprise transactions (Sorbet) was developed to provide

- secure sessions
- lightweight transactions for transferring large blobs of information
- client-side authentication
- comparable performance to SSL

Although initially developed for common object request broker architecture (CORBA), different flavors of Sorbet could be easily developed to work with other distributed object technologies such as JavaSoft's remote method invocation (RMI) or Microsoft's distributed component object model (DCOM). The remainder of this paper describes an implementation of Sorbet as well as providing a comparison with Inprise's SSL Pack.

[1] VisiBroker for Java SSL Pack uses HTTPS, a security mechanism built into Netscape and Microsoft browsers. By using these facilities of the browser, an applet does not need any special understanding of SSL, eliminating the need to pre-install classes or native libraries on the client.

2 Background

Security can mean different things, depending on the type of system, the type of data, and associated risks. This section provides background on some techniques for securing systems and data including confidentiality; identification and authentication; data integrity; and secure sessions.

2.1 Confidentiality

Confidentiality prevents third parties from viewing information sent between two communicating parties. Perhaps the most widely used method of providing confidentiality over an insecure medium is the use of cryptography. Cryptography involves the sender transforming, or encrypting, the message in such a way that the message cannot be read en route. The message is then reconstructed, or decrypted, into its original form by the receiver. There are many modern cryptographic techniques [Schneider 95]. These are divided into two classes, symmetric and asymmetric (also called public/private) key cryptography. The basic ideas are described here and the two classes are explored more fully afterwards.

Cryptography is an algorithmic process of converting a plain-text (or clear-text) message to a cipher (or cipher-text) message based on an algorithm that both the sender and receiver know, so that the cipher-text message can be returned to its original, plain-text form. There are a number of algorithms for performing this process, but there are comparatively few such algorithms that, having stood the test of time, prevent the plain-text from being revealed by someone other than intended reader. Most such algorithms require the use of a *key*. A key is simply a parameter to the algorithm that permits the proper transformation. In symmetric key cryptography, the same key is used for enciphering and deciphering. In asymmetric key cryptography, one key is used for enciphering and another, mathematically related key, is used for deciphering.

Symmetric Key Cryptography

Perhaps the most widely used symmetric key cryptographic method is the Data Encryption Standard (DES) [DES 93]. Although published in 1977 by the National Bureau of Standards, DES has not yet been replaced. The original publication is reprinted in [Beker 82]. DES uses a fixed length, 56-bit key. Its advantage is that it is an efficient algorithm, and can be processed extremely fast particularly if special hardware is used. By increasing the key size, DES becomes even more secure. A variation of DES, called Triple-DES or DES-EDE (encrypt-decrypt-encrypt), uses three applications of DES and two, independent DES keys to produce an effective key length of 168 bits [ANSI 85].

The International Data Encryption Algorithm (IDEA) was invented by James Massey and Xuejia Lai of ETH Zurich, in Switzerland, in 1991 and is patented and registered by the Swiss Ascom Tech AG, Solothurn [Lai 92]. IDEA uses a fixed length, 128-bit key, but it is faster than Triple-DES. Also in the early 1990s, Don Rivest of RSA Data Security, Inc., invented the algorithms RC2 and RC4. These use variable length keys and are claimed to be faster even than IDEA; however, implementations may be exported from the U.S. only if they use key lengths of 40 bits or less.

Although symmetric key cryptography can be very secure, it has a fundamental problem—key management. Since the same key is used for encryption and decryption, it must be kept secure. If an adversary knows the key then the message can be decrypted. However, the key must be available to the sender and the receiver and these two parties may be physically separated. Symmetric key cryptography transforms the problem of transmitting messages securely into that of transmitting keys securely. This is a step forward, because keys are much smaller than messages, and the keys can be generated beforehand. Nevertheless, the problem of ensuring that the sender and receiver are using the same key and that potential adversaries do not know this key is a major stumbling block in the use of symmetric key cryptography. This is referred to as the key management problem.

Public/Private Key Cryptography

Asymmetric key cryptography overcomes the key management problem because the encryption and decryption keys are no longer the same. Algorithms are available for which the key used for decryption cannot be determined with reasonable effort even when the key used for encryption is known. Therefore, the encryption key can be made public, provided the decryption key is kept private to the party wishing to receive encrypted messages (hence the name public/private key cryptography). Now, anyone can use the public key to encrypt a message to be sent to the recipient, but only the recipient can decrypt the message.

James Ellis, Malcolm Williamson, and Clifford Cocks first investigated public/private key cryptography at the British Government Communications Headquarters (GCHQ) in the early 1970s. However, at the time they kept their findings secret. The papers have now been published [Ellis 87]. The first public discussion of public/private key cryptography was by Whitfield Diffie and Martin Hellman in 1976 [Diffie 76]. The Diffie-Hellman algorithm is now used to establish the secret key for a symmetric key system, but it is not efficient enough to be used for the encryption of large messages.

A widely used public/private key system is RSA, named after the initials of its inventors, Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman [PKCS 91]. RSA depends on the difficulty of factorizing the product of two very large prime numbers. Although used for encrypting whole messages, RSA is much less efficient than symmetric key algorithms such as DES. ElGamal is another public/private key algorithm [El Gamal 85]. It depends on an arithmetic algorithm different from that of RSA, the so-called discrete logarithm problem.

An extensive discussion of public/private key cryptography, including much of the mathematical detail, can be found in the book, *Public Key Cryptography* [Salomaa 96].

2.2 Identification and Authentication

Although the keys used in public/private key cryptography are different, the keys are mathematically related in that a message encrypted with either key can be decrypted with the other. For confidentiality, messages must be encrypted with the public key so that only the holder of the private key can decrypt them. However, there is an application for doing things the other way round. If a message is encrypted with someone's private key, that person's public key will decrypt it. Hence, recipients can be absolutely certain that the message originated from the correct person because only they could have encrypted it. This leads to the idea of authentication. A service wishing to be certain that it is communicating with who it thinks it is can ask the communicating party to send a message encrypted with their private key. If the message successfully decrypts with the communicating party's public key then the service can be sure that the other party is genuine.

This begs the question of the ownership of public keys. This problem is addressed through the use of certificates. Certificates bind an identity to a public key by way of a third party acting as a certificate authority. Confidence that the certificate actually represents an identity is based on trust in the assurance mechanisms implemented by the certificate authority.

2.3 Data Integrity

An extension of the idea of authentication enables us to ensure data integrity. For example, the sender creates a secure digest of the original data (by using a secure hash function, such as MD5 [Rivest 91] or SHA-1 [SHA 95]) and includes a copy of this digest, encrypted with the sender's private key, with the data. The recipient can then decrypt the digest using the sender's public key and check that the data still hashes to the same value. Nobody other than the original sender can produce the encrypted digest because only the sender has access to the private key. Hence, the integrity of the data is ensured.

Data integrity can also be achieved by means of encryption, as most decryption algorithms would fail if the integrity of the data has been comprised. Using encryption to provide data integrity is overkill, unless confidentiality is also a requirement.

2.4 Secure Sessions

Hijacking is a security threat where an adversary takes over an existing session without the knowledge of one or both parties. This technique allows the hijacker to bypass the authentication process but still access private data or perform restricted operations. One way of preventing a client/server session from being hijacked is to use a session key and cookies. The client authenticates itself with the security server, as described above. The security server then provides a session key that is used by the client to identify itself for the session. This session key can be smaller than the full credentials used by the client to identify it initially. With each client/server call, the client provides the session key and a cookie, the cookie being unique for each call. The server checks that the session key and cookie match.

A straightforward means of implementing this session key and cookie approach is to use a pseudo-random number generator (PRNG). Instances of the PRNG are started on both the client and server sides, seeded by the session key. The cookie is the next random number provided by the PRNG as each client call is being prepared. The server simply checks that the cookie provided by the client and the next random number produced by the appropriate PRNG, as identified by using the session key, is the same. Note that the server will have one PRNG running for each client it is currently serving.

To be secure, the session key must be passed to the client encrypted, but this can be done using a public/private key algorithm, since the security server has received the client's public key as part of the initial authorization process. The random number generator must also be secure so that it is less subject to pattern matching and other attacks that can be made on traditional random number generators. Cryptographically secure pseudo-random number generators are available [ECS 94].

This session key and cookies technique is secure and reasonably "lightweight." The session key and cookies do not need to be very large. Even if an adversary were able to generate the correct next random number and interpose itself in the client/server stream, the random number sequence would immediately get out of step and the server could close the connection. Hence, any compromise would be limited.

3 Secure Sessions Using CORBA Interceptors

CORBA (Common Object Request Broker Architecture) [OMG 98] is a popular standard used for developing distributed applications. With CORBA, developers can use interceptors to remove much of the security code from the body of the implementation. Interceptors are an optional extension to the ORB [Inter 98]. An interceptor is interposed in the invocation and response paths between a client and its server object and can inspect and modify messages as they travel between the client and server.

An experimental implementation of Sorbet was developed using the Inprise VisiBroker CORBA implementation on the Java 2 platform. The VisiBroker implementation allows for three types of interceptors to be installed: bind, client, and server. Bind interceptors are aware of the messages generated when a client binds to a server. Client interceptors are associated with a client object, and are aware of the messages generated when a client makes a call to a server, and of the response to those messages from the server. Server interceptors sit at the other end of the communications link, and are aware of the messages received by a server, and of the server's response to these messages. Sorbet uses client and server interceptors.

A sample application was developed using Sorbet, consisting of a CORBA application service, a security service and a simple client. Client interceptors authenticate the client to a security service. The security service is a session object factory that creates session objects for authenticated clients. After successfully authenticating the client, a session object is returned to the client interceptor. This session object teams with the client and server interceptors to monitor and enforce session security between the client and application service. These operations are detailed in the following sections.

3.1 Client Authentication and Session Initialization

The sample Sorbet application consists of a client, application server, and security server as shown in Figure 1.

At startup, the client and server each register interceptor factories. The client then makes an initial call to the application server. This call causes the client interceptor factory to create a client interceptor. The client interceptor initiates a series of steps to authenticate to the security service and receive a valid session object, as shown in Figure 2 and described below. Bracketed numbers correspond to numbered transactions in the figure.

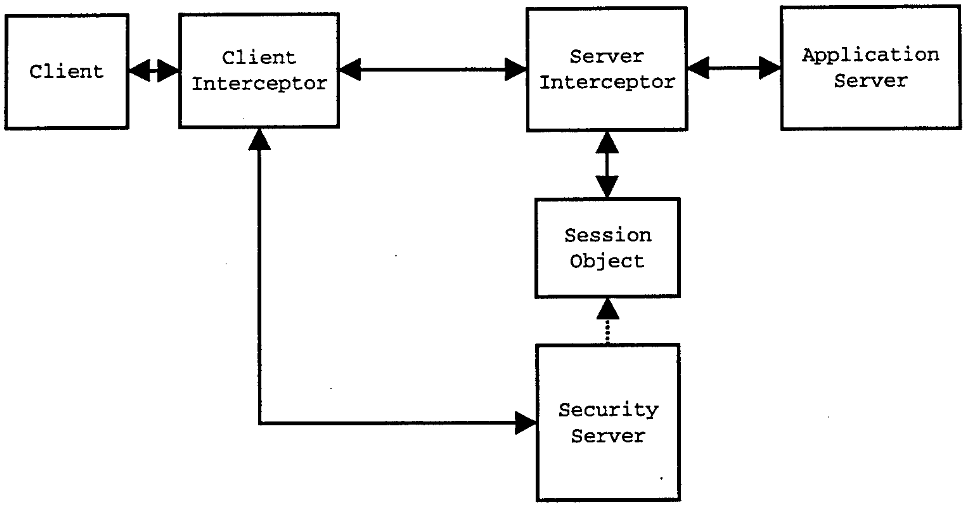


Figure 1. Secured Application Using Sorbet

Initially, the client interceptor loads the credentials from a keystore[2] located on the client platform and then presents these to the security service [1]. The security service creates a challenge consisting of a random stream of bytes and returns it to the client interceptor [2,3]. The client interceptor formulates a response to the challenge by signing the challenge with the user's private key [4]. The client interceptor calls the security service to retrieve the seed (generated by a cryptographically secure PRNG) which is then held by the client.

The client interceptor passes the response to the security service to create a session [5]. The security service uses the client's public key to verify that the response received was in fact the random data signed using the client's private key [6]. If the response is verified, the security service creates a session object using the negotiated seed [7]. An object reference to the newly created session object is returned to the client interceptor [8]. The session is set as the default principal on the ORB [9]. If the response sent to the security service fails the verification test, no session is created and an exception is thrown on the client.

Assuming client authentication is successful, both the session object and the client now use the same seed to start tandem PRNGs to provide a synchronous step operation. The session object returns the first of a sequence of reproducible random numbers encrypted using the client's public key [10].

[2] A database of private keys and their associated X.509 certificate chains.

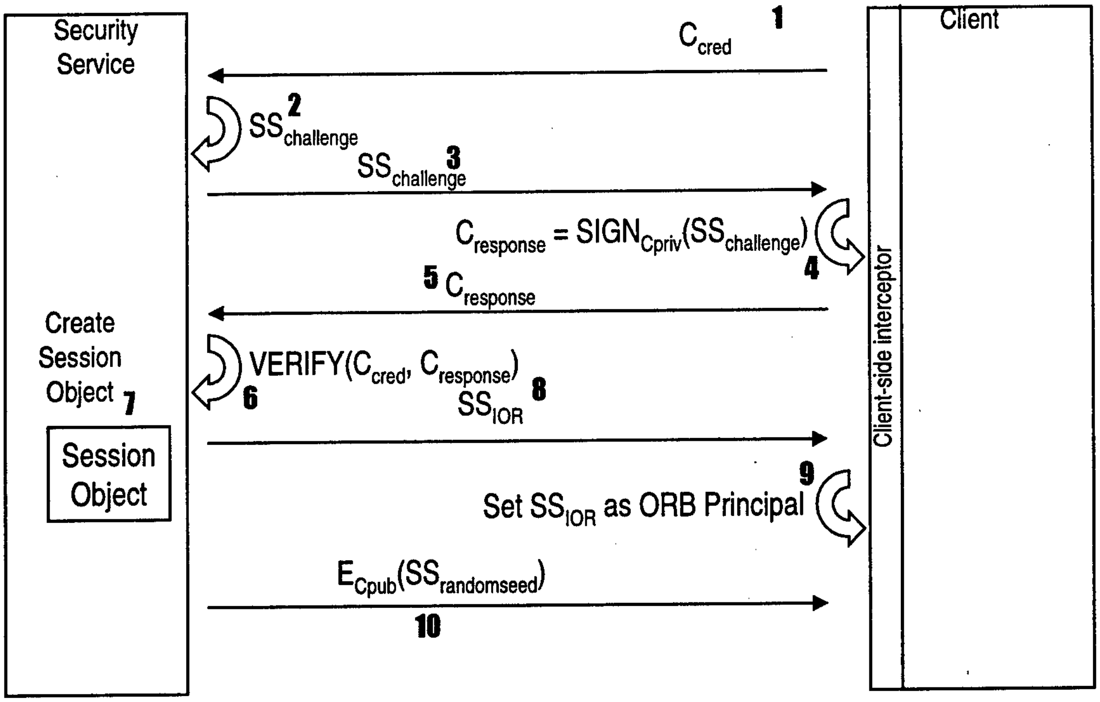


Figure 2. Client Authentication

3.2 Sorbet Secure Session

Once the client interceptor is initialized, the client can begin making calls to the application service. The first call to the application service causes the server interceptor factory to create a sever interceptor. The server interceptor, client interceptor, and session object work together to provide a secure session, as described in Figure 3.

For each call made by the client, the client interceptor *steps* the client's PRNG and attaches the random number obtained to the message [1]. The application server interceptor takes the random number off the message and asks the session object (which it identifies from the principal) if the random number is correct [2,3]. When it receives this request, the session object steps its PRNG and compares the number produced with the number extracted from the message [4]. The session object returns a result of true or false to the server interceptor, based on this PRNG comparison. If the check passes, the server interceptor allows the message (with the random number removed) through to the application server [5]. If the check fails, the server interceptor throws an exception that is caught by the client. Further action could be taken on the application server side to log the attempt, break the connection, or whatever action is required by the security policy. The final result of the application server call is returned to the client as a normal IIOP reply [6].

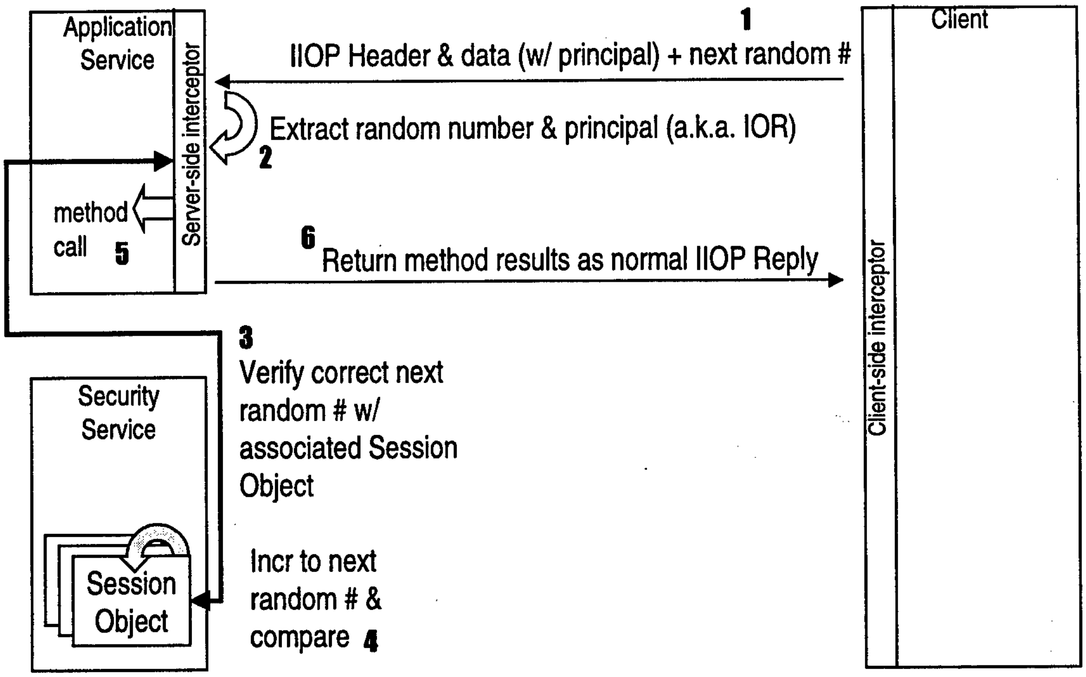

Application Service
Server-side interceptor
1
IIOP Header & data (w/ principal) + next random #
Client
Extract random number & principal (a.k.a. IOR)
2
method call 5
6
Return method results as normal IIOP Reply
Client-side interceptor
3
Verify correct next random # w/ associated Session Object
Security Service
Session Object
Incr to next random # & compare 4

Figure 3. Secure Session Operation

Optimized Model

The Sorbet model is designed to support a distributed object system where distributed objects can freely interact with other objects in the system. As a simplification of this model, we have assumed that one of these objects is a client that may operate outside the secure computing base, but that this client object may communicate with multiple back-end servers to perform various system functions.

Recall from the previous section that Sorbet uses a synchronous step operation to ensure that a secure session is maintained. The client steps the function for each remote invocation of a server's methods. In a pure client-server scenario, the client and the server are both capable of keeping this step function synchronized. However, when multiple servers are used, each server has no way of knowing when another server has been invoked. Invoking a second server would instantly cause the step function to become unsynchronized, resulting in a security violation.

To resolve this problem, a separate session object was to provide a single coordination point for stepping the server-side function. However, this approach has the drawback of requiring an additional remote invocation from the server interceptor to the session object for each data packet. If, for example, the security comparison of the two PRNGs is performed in the server interceptor and not the session object, the extra session object call would not be required. Although we have not implemented this solution, we have experimented with the performance that would result from eliminating this extra remote method invocation as our optimized model. The results of these experiments are discussed in the Comparison section of this paper.

Distributed Object Systems

The greatest threat to information often comes from within an Intranet. The threat may come from multiple vendors working at a single customer location, customers working at a vendor location, or simply a disgruntled employee. In any case, system operations within an Intranet may need to be protected as much as or more than Internet transactions.

One way to do this is to extend the security model to operate as a true distributed object system. In this model, the notion of a client and or server process is eliminated. Instead, each process may operate as both a server and a client to other processes in the system.

To secure this system, each process could be assigned a public/private key pair as a secure means of identification. Each process could also maintain a list of registered processes, represented by their digital certificates (containing public keys). Each session established between servers would then be authenticated and secured using the session key and cookies method used by Sorbet. This solution would build upon the optimized model described in the previous section, where each process maintains a hash of current sessions along with a corresponding session object (implemented as a class or data structure in the same process space). The session key is used as a hash key to retrieve the session object, allowing the correct PRNG to be stepped.

3.3 Separation of Security Policy From Core Application

It is important to separate the core client/server software from that involved in providing the security, since both parts of the software might need to be updated or replaced independently of the other. One advantage of Sorbet is that the application server is completely unaware of the security mechanisms that are in place. The client must initialize the client interceptor by passing a keystore alias, keystore password, and a reference to the ORB. Apart from this, the client takes no part in the security mechanisms—the interceptors do all the work of the security mechanism. However, implementation of interceptors is still vendor specific and the CORBA interceptor specification may itself change. (An RFP for portable interceptors has been issued [RFP 98].)

4 Comparison

This section compares five different security models: unsecured, full SSL, SSL without encryption (SSL null), Sorbet, and a theoretical "optimal" model. The unsecured model transfers data over IIOP without any form of security. The full SSL solution implements secure sessions, client and server authentication, and encryption for confidentiality. SSL null eliminates encryption (therefore, no confidentiality) but retains secure sessions and client server authentication. The Sorbet implementation, explained in detail in the previous section, supports secure sessions and client authentication, but not confidentiality. The theoretical "optimal" model assumes that the Sorbet implementation can be implemented in such a way as to avoid an extra remote invocation on the session object.

4.1 Data Transfer Rate

Performance was measured using the different security models by transmitting a 2Mb file over IIOP in 0.5Kb, 1Kb, 10Kb and 100Kb fragments. Over 4000 transactions are required to move the file using 0.5Kb fragments; 20 transactions are required using 100Kb fragments.

As illustrated in Table 1, each security model has unique performance characteristics. Unsecured data transfer has the best performance, with measured data transfer rates of 159.9 bytes/second using 0.5K segments compared to 82.8 for full SSL with same packet size. SSL performance degrades significantly as the size of the IIOP message increases. In fact, CPU intensive encryption calculations for each packet result in performance limitations imposed by CPU capacity.

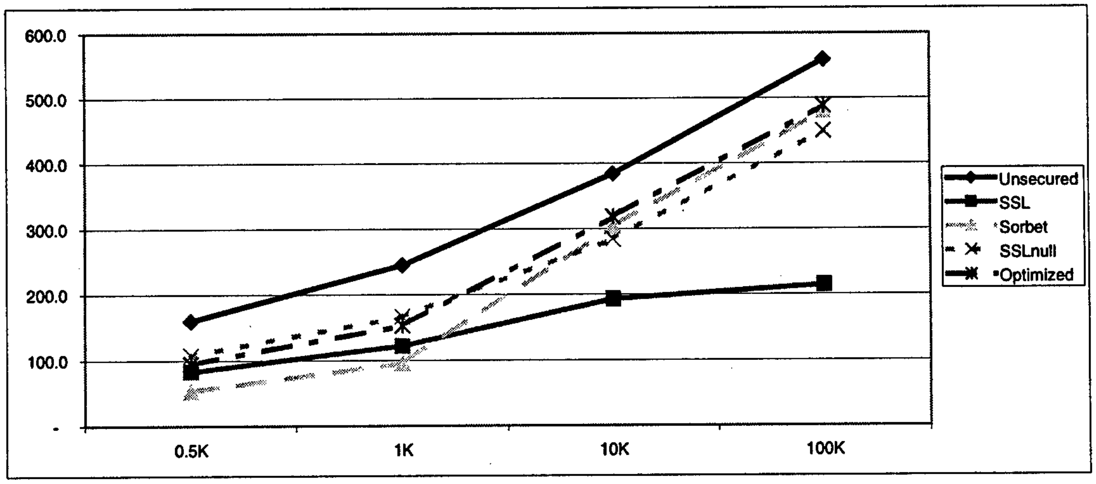


Table 1. Data Transfer Rate vs. IIOP Packet Size

Sorbet has poorer performance than all other security models at 0.5k data packet sizes. This is a result of the extra remote object method invocation on the session object. This indicates that Sorbet, as implemented, is a poor choice for distributed object systems primarily consisting of many small messages. At larger packet sizes, Sorbet competes closely with SSL without encryption – showing that Sorbet is a viable alternative to SSL for the secure transfer of large packets when confidentiality and data integrity are not requirements.

4.2 Digital Signing and Verification Performance

All measures for Sorbet taken in the previous section used Sun Java Cryptologic Extension (JCE) technology for signing and verification. Signing and verification in Sorbet takes place during initialization of a connection. For example, in a CORBA-based application this initialization occurs during the `bind()` operation. The time it takes to sign and verify can vary significantly (see Table 2 and Table 3) based on the product used to perform the cryptologic functions and the platform.

	Solaris 2.5.1 (first)	Solaris 2.6 (first)	WinNT (first)	Solaris 2.5.1 (average)	Solaris 2.6 (average)	WinNT (average)
Crypto-J	2.4	2.1	2.1	7.7	4.9	0.08
SunJCE	16.1	9.9	5.5	1.3	0.9	0.08

Table 2. Signing Performance

Both signing and verification measures were taken on three platforms (Solaris 2.5.1, Solaris 2.6 and WinNT). Data was collected for the first time the function was invoked and an average of the subsequent four invocations. This was important in that, for some operations, there was greater than a magnitude difference in performance between the first and subsequent calls.

	Solaris 2.5.1 (first)	Solaris 2.6 (first)	WinNT (first)	Solaris 2.5.1 (average)	Solaris 2.6 (average)	WinNT (average)
RSA	3.2	2.5	2.0	1.5	0.9	0.1
Sun	0.3	0.2	0.2	0.	0.1	0.1

Table 3. Verification Performance

The Crypto-J package had much better times than SunJCE for signing, but verification times were better in SunJCE. If we replaced the use of SunJCE with Crypto-J in Sorbet, we would expect to see an overall decrease in the time spent in initialization, which could improve throughput in systems with many connections.

5 Conclusions

Commercial availability of distributed application infrastructures such as CORBA provides application developers a framework to add services, such as security policies, independently of the core client/server software. SSL provides a standard, commercial solution for securing transactions between a client and server in an Intranet environment or over the Internet. Current implementations of SSL have some limitations in that they often rely on native libraries.

Sorbet does not require native libraries, providing a portable solution that can be easily deployed with a broad range of client architectures. In addition, Sorbet provides application developers more control over security policy. Table 1 shows that Sorbet can be just as cost effective as an SSL solution providing similar functionality. Also, Sorbet allows the security policy to be separated from the core application, which allows for flexibility when the software is being maintained or updated.

In most cases, SSL is a better choice than a custom security model such as Sorbet because SSL is a standard solution that can be more readily approved for use in large organizations. Custom solutions such as Sorbet may be used as a last resort when COTS solutions prove inadequate due to performance, functionality, or other failures.

References

ANSI 85 ANSI X9.17-1985, *American National Standard, Financial Institution Key Management (Wholesale)*, American Bankers Association, Section 7.2. New York: American National Standards Institute, 1985.

Beker 82 Beker, H. & Piper, F. *Cipher Systems*. London: Northwood Books, 1982.

DES 93 *Data Encryption Standard (DES)* (FIPS PUB 46-2). Gaithersburg, Md.: National Institute of Standards and Technology, January, 1993. Available WWW: <URL: http://www.nist.gov/itl/div897/pubs/fip46-2.htm>.

Diffie 76 Diffie, W. & Hellman, M.E. "New Directions in Cryptography." *IEEE Transactions on Information Theory*, IT-22, Vol. 6, pp. 644–654, 1976.

ECS 94 Eastlake, D., Crocker, S. & Schiller, J. "Randomness Recommendations for Security." RFC 1750, Network Working Group, IETF, December, 1994.

El Gamal 85 El Gamal, T. "A Public Key Cryptosystem and Signature Scheme Based on Discrete Logarithms." *IEEE Transactions on Information Theory*, IT-31, pp. 469–473, 1985.

Ellis 87 Ellis, J.H. "The Story of Non-Secret Encryption." Cheltenham, UK: Communications Electronics Security Group, 1987. Available WWW: <URL: http://www.cesg.gov.uk/about/nsecret/ellis.htm>.

Inter 98 Object Management Group. Ch. 18, "Interceptors." *CORBA 2.2 Specification* (OMG 98-07-01). Framingham, Ma.: Object Management Group, 1998. Available WWW: <URL: ftp://www.omg.org/pub/docs/formal/98-02-23.pdf>.

Lai 92 Lai, X. *ETH Series on Information Processing* (J.L. Massey, ed.). Vol. 1, *On the Design and Security of Block Ciphers*. Konstanz, Switzerland: Hartung-Gorre Verlag, 1992.

OMG 98 Object Management Group. *CORBA 2.2 Specification* (OMG 98-07-01). Framingham, Ma.: Object Management Group, 1998. Available WWW: <URL: http://www.omg.org/library/c2indx.html>.

PKCS 91 PKCS #1: *RSA Encryption Standard*, Version 1.4. San Mateo, Ca.: RSA Data Security, Inc., 1991.

RFP 98 Object Management Group. *Portable Interceptors RFP*, Draft 6 (orbos/98-09-05). Framingham, Ma.: Object Mangement Group, 1998. Available WWW: <URL: http://www.omg.org/docs/orbos/98-09-05.pdf>.

Rivest 91 Rivest, R. *The MD5 Message Digest Algorithm*. Cambridge, Ma.: MIT Laboratory for Computer Science, 1991.

Seacord 98 Seacord, R. C. & Hissam, S. "Browsers for Distributed Systems: Universal Paradigm or Siren's Song?" *World Wide Web Journal 1*, 4, Baltzer Science Publishers BV, 1998.

SHA 95 *Secure Hash Standard* (FIPS 180-1). Gaithersburg, Md.: National Institute of Standards and Technology, April 1995. Available WWW: <URL: http://www.nist.gov/itl/div897/pubs/fip180-1.htm>.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. **AGENCY USE ONLY** (LEAVE BLANK)	2. **REPORT DATE** July 1999	3. **REPORT TYPE AND DATES COVERED** Final

4. **TITLE AND SUBTITLE** Securing Internet Sessions With Sorbet	5. **FUNDING NUMBERS** C — F19628-95-C-0003
6. **AUTHOR(S)** Fred Long, Scott Hissam, Robert C. Seacord, John Robert	
7. **PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	8. **PERFORMING ORGANIZATION REPORT NUMBER** CMU/SEI-99-TN-002
9. **SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)** HQ ESC/DIB 5 Eglin Street Hanscom AFB, MA 01731-2116	10. **SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. **SUPPLEMENTARY NOTES**

12.A **DISTRIBUTION/AVAILABILITY STATEMENT** Unclassified/Unlimited, DTIC, NTIS	12.B **DISTRIBUTION CODE**

13. **ABSTRACT** (MAXIMUM 200 WORDS)

More and more organizations are using intranets, and even the Internet, as the communications media for important data. However, such communications media are inherently insecure and subject to hijacking. To secure these connections, mechanisms must be built on top of the underlying communications facilities. In this paper, we discuss one such security mechanism and describe an implementation using common object request broker architecture (CORBA) -based interceptors.

14. **SUBJECT TERMS** common object request broker architecture (CORBA), secure socket layer (SSL), sorbet	15. **NUMBER OF PAGES** 15 pp.
	16. **PRICE CODE**

17. **SECURITY CLASSIFICATION OF REPORT** UNCLASSIFIED	18. **SECURITY CLASSIFICATION OF THIS PAGE** UNCLASSIFIED	19. **SECURITY CLASSIFICATION OF ABSTRACT** UNCLASSIFIED	20. **LIMITATION OF ABSTRACT** UL